Holistically Budgeting Processing Graphs

Zelin Tong, Shareef Ahmed, and James H. Anderson
Department of Computer Science
University of North Carolina at Chapel Hill

***Abstract*—To certify the schedulability of a system, valid per-task worst-case execution-time (WCET) estimates are almost always required. Unfortunately, on multicore machines, deriving WCET estimates through static analysis that is not highly pessimistic may never be a practical reality. The alternative is to determine WCETs via a measurement process, but such a process cannot correctly produce accurate WCET estimates with certainty. This lack of certainty necessitates the use of overrun-handling mechanisms, such as budget-enforcement techniques, to preserve temporal correctness at runtime. In many systems of interest today, tasks are interconnected to form processing graphs, which can be quite large. The simplest (and perhaps most common) approach to budget enforcement in this case is to abort an entire graph invocation whenever any node (task) overruns its budget. However, such an approach can result in a high abort rate at the graph level even when the per-node abort rate is low. To remedy this situation, this paper presents a holistic budget-management strategy for directed acyclic graphs (DAGs) that involves reallocating per-node budgets to overrunning nodes to avoid DAG-level aborts. To enable the effects of aborts to be studied analytically, a probabilistic analysis is presented to derive a DAG's abort rate under the proposed budget-management strategy. Experimental results are also presented to demonstrate the utility of budgeting graphs holistically.**

I. Introduction

Today, artificial-intelligence (AI) techniques are fueling the realization of ever more sophisticated embedded systems, with autonomous vehicles being a prime example. These techniques typically involve executing tasks with dataflow dependencies as expressed in the form of a processing graph. These graphs can be large and computationally demanding, necessitating the use of a multicore machine. In a safety-critical context such as an autonomous vehicle, these graph-based workloads must pass *real-time safety certification*, which entails performing both timing analysis and schedulability analysis. The goal of *timing analysis* is to produce *worst-case execution times* (WCETs) of executable code; *schedulability analysis* in turn validates timing constraints, assuming WCETs are provided.

Multicore timing-analysis dilemma. Unfortunately, in the context of multicore machines, the coupling of timing analysis and schedulability analysis creates a dilemma [38]. Due to the highly complex nature of multicore machines, there is consensus today that *static* timing analysis, which computes the WCET of a program by analyzing its code structure, may never be a practical solution for such machines [42]. The alternative is *measurement-based* timing analysis, which may not capture the true WCET of a program. Thus, *the WCET values that schedulability analysis relies upon can never be assumed to be correct with certainty.*

Work was supported by NSF grants CPS 1837337, CPS 2038855, CPS 2038960, and CNS 2151829, ARO grant W911NF-20-1-0237, and ONR grant N00014-20-1-2698.

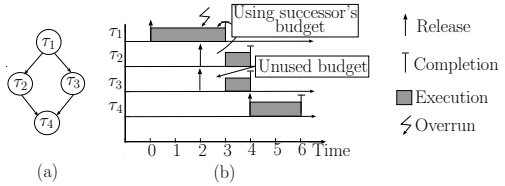


Fig. 1: (a) A DAG task, and (b) executing an overrunning job using successor node's budget.

Mitigating this dilemma by budget enforcement. A system's timing correctness may be violated at runtime if the WCETs assumed in schedulability analysis are overrun. A common approach to addressing this problem is by enforcing per-task execution budgets: each task is assigned a budget based on its WCET, and if an invocation of that task (*i.e.*, one of its jobs) overruns said budget, it is aborted.

The need to budget graphs holistically. When tasks are embedded as nodes in a processing graph, an entire graph invocation typically becomes logically incorrect when one of its node invocations is aborted, so the entire graph invocation should be aborted in such a case. However, such an approach can cause the effects of node WCET overruns to be greatly magnified. For example, a node failure rate of only 0.1% can be magnified to a graph failure rate of 36.8% (assuming IID probabilities) for a graph with 1,000 nodes [2]—in AI-based use cases, graphs of this size are a definite possibility.

These observations motivate the need for more holistic ways of budgeting graphs that are more flexible, particularly in these new use cases. This need has been highlighted by other researchers in the context of ROS-based robotic systems [8] and component-based accelerator-using systems [2]. In this paper, we consider this need in the context of budgeting computations expressed as directed acyclic graphs (DAGs).

Avoiding node-invocation aborts. The frequency of DAG-level aborts can be reduced by adopting a holistic budget-management strategy that allows *budget-reallocation* within a DAG. Using such a strategy, an overrunning node invocation can be executed longer by consuming budgets of one or more invocations of "downstream" nodes. However, this approach is subject to nuances due to data dependencies. For example, consider the DAG shown in Fig. 1(a) and a schedule of the DAG in Fig. 1(b). Task τ_1's invocation in Fig. 1(b) overruns

2576-3172/23/$31.00 ©2023 IEEE
DOI 10.1109/RTSS59052.2023.00013

its budget at time 2. It is allowed to execute during the time interval $[2, 3)$ using the budget of task τ_2's invocation, which causes an additional delay in task τ_3's invocation too. Thus, reallocating budgets within a DAG requires carefully sifting through the various reallocation possibilities that exist and their concomitant data-dependency side effects.

Contributions. In this paper, we provide a foundation for graph-level budgeting through three contributions.

First, we propose a holistic budget-management policy for DAG tasks that includes budget- and slack-reallocation within a DAG. Our budget-reallocation technique allows an overrunning node invocation to execute on invocations of "downstream" nodes. Moreover, our slack-reallocation method reduces the likelihood of overruns by allowing the slack produced by an underrunning node invocation to be consumed by other node invocations. We also detail the case when aborting an entire DAG invocation is inevitable to prevent runtime violations of timing guarantees. Our proposed budget-management policy is agnostic to the DAG scheduling strategy.

Second, we provide an analysis to upper bound the probability of DAG aborts when our budget-management policy is used. Our analysis probabilistically quantifies an overrunning node invocation's impact on other nodes. Moreover, our analysis handles complicated dependencies that may arise from precedence constraints in the presence of budget overruns.

Third, we present results from experimental evaluations that demonstrate the improvement of the DAG abort rate compared to the naïve approach of aborting an entire DAG invocation when a node overrun occurs. Our experiments show that a DAG's abort rate can be improved significantly by adopting our proposed budget-management strategy.

Organization. In the rest of this paper, we provide necessary background information (Sec. II), present budgeting mechanisms for DAGs and corresponding drop-rate analysis (Secs. III and IV), present our experimental results (Sec. V), review related work (Sec. VI), and conclude (Sec. VII).

II. Background

We consider a task system Γ consisting of N DAG tasks scheduled on m identical processors.

Each *DAG task* $G \in \Gamma$ is characterized by (V, E), where V is a set of n nodes representing n *tasks* $\tau_1, \tau_2, \ldots, \tau_n$, and E is a set of directed edges. (We use the terms "task" and "node" interchangeably.) An edge from τ_i to τ_k specifies a precedence constraint between the *predecessor* task τ_i and *successor* task τ_k. The set of predecessors (resp., successors) of τ_i is denoted by $pred(\tau_i)$ (resp., $succ(\tau_i)$). If a path exists from τ_i to τ_k, then τ_i (resp., τ_k) is called an *ancestor* (resp., *descendant*) of τ_k (resp., τ_i). The set of ancestors of τ_i is denoted as $anc(\tau_i)$. We assume that τ_1 (resp., τ_n) is a unique source (resp., sink) task for the DAG G with no incoming (resp., outgoing) edges. Multiple sources/sinks can be supported by adding "virtual" sources/sinks with a WCET of zero.

Each task τ_i releases a sequence of *jobs* $J_{i,1}, J_{i,2}, \ldots$. The source task τ_1 releases jobs sporadically, *i.e.*, its releases have a minimum separation given by G's *period*, denoted by T. The j^{th} job of a non-source task τ_i is released once the j^{th} jobs of all of its predecessors have completed execution. The *release time* (resp. *finish time*) of $J_{i,j}$ is denoted by $r_{i,j}$ (resp., $f_{i,j}$). The *response time* of G is $\max_j\{f_{n,j} - r_{1,j}\}$.

Task τ_i has an execution budget of C_i, which equals its estimated WCET, and a *probabilistic WCET* (pWCET) [9] e_i, which is a discrete random variable (RV) representing the actual execution times of its jobs. Each task τ_i is also given a relative deadline D_i. We do not require deadlines to be hard, *i.e.*, deadline misses are acceptable. We assume time to be discrete and a unit of time to be 1.0. Tbl. I summarizes our notation.

TABLE I: Task-model notation summary.

Symbol	Meaning	Symbol	Meaning
N	No. of DAG tasks	C_i	Execution budget of τ_i
m	No. of processors	e_i	pWCET of τ_i
Γ	Task system	$r_{i,j}$	Release time of $J_{i,j}$
G	A DAG task	$f_{i,j}$	Finish time of $J_{i,j}$
n	No. of nodes in G	S_i	Reservation server of τ_i
V	Set of nodes in G	$S_{i,j}$	j^{th} job of S_i^ℓ
E	Set of edges in G	$pred(\tau_i)$	Set of predecessors of τ_i
τ_i	i^{th} task of G	$succ(\tau_i)$	Set of successors of τ_i
$J_{i,j}$	j^{th} job of τ_i	$anc(\tau_i)$	Set of ancestor of τ_i
T	Period of G	$dep(J_{i,i})$	Set of jobs on which $J_{i,j}$ depends
D_i	Rel. deadline of τ_i		

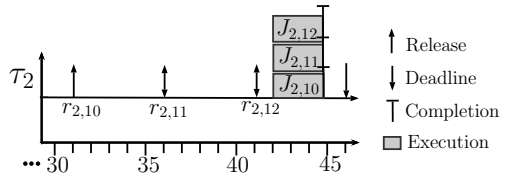


Fig. 2: Illustration of intra-task parallelism for τ_2.

Intra-task parallelism. We allow consecutive jobs of the same task to execute in parallel. Allowing such intra-task parallelism may decrease *analytically derived* response-time bounds [15]. Moreover, intra-task parallelism enables task *utilizations*, *i.e.*, C_i/T, to exceed 1.0. Thus, since a job of a task does not depend on prior jobs of the same task, dependencies among jobs of different tasks can be defined as follows.

Def. 1. *A job $J_{i,j}$* depends on *job $J_{k,j}$ if $\tau_k \in anc(\tau_i)$ holds. We let $dep(J_{i,j})$ denote the set of jobs on which $J_{i,j}$ depends.*

Ex. 1. *Assume that G in Fig. 1(a) is scheduled on $m = 3$ processors. Fig. 2 depicts a possible schedule of jobs $J_{2,10}, J_{2,11}$, and $J_{2,12}$, which come from three successive invocations of G. None of the three jobs executes before time 42. Assume that all jobs with higher priority than these three jobs complete execution before time 42. Since intra-task parallelism is allowed, all three jobs are scheduled at time 42. Note that intra-task parallelism is common in AI applications. For example, G here might be a computer-vision objection-detection application that can process successive video frames simultaneously.*

Working with probabilities. A discrete RV X can take on an integer value x with probability $P(X = x)$. X is

characterized by a *probability distribution function* (PDF), f_X, where $f_X(x) = P(X = x)$. The *cumulative distribution function* (CDF) of X, denoted as F_X, is defined as $F_X(x) = \sum_{i=-\infty}^{x} f_X(i)$ and is equal to $P(X \leq x)$. The CDF's compliment, the *complementary CDF* (CCDF) of X, denoted as $\overline{F}_X$, is defined as $\overline{F}_X(x) = 1 - F_X(x)$ and is equal to $P(X > x)$.

Given two RVs X_1 and X_2, $P(X_1 > x_1, X_2 > x_2)$ denotes the probability of $X_1 > x_1$ and $X_2 > x_2$ and the *joint* PDF of X_1 and X_2 is given by $f_{X_1,X_2}(x_1, x_2) = P(X_1 = x_1, X_2 = x_2)$ for any values x_1 and x_2. Also, $P(X_1 > x_1 \mid X_2 = x_2)$ denotes the probability of $X_1 > x_1$ given $X_2 = x_2$.

Comparison between RVs. There exist multiple ways in which RVs can be ordered. For RVs X and Y, X is *state-wise* at least Y, denoted as $X \geq_0 Y$, when the observed value of X is at least the observed value of Y. Alternatively, X is *first-degree* at least Y, denoted as $X \geq_1 Y$, when $P(X > x) \geq P(Y > x)$ for all values of x. It is known that the former implies the latter [37].

III. Budgeting DAG-based Task System

In this section, we propose several budget-management policies for DAG tasks. We begin by defining per-node reservation servers that act as containers within which jobs execute.

A. Reservation of a Task

Consider a DAG task $G \in \Gamma$. For each node τ_i, we define a *reservation server* S_i with budget C_i and relative deadline D_i. Each server releases server jobs. We denote the j^{th} *server job* of S_i by $S_{i,j}$, which corresponds to job $J_{i,j}$. We call the server job $S_{i,j}$ the *reservation* of job $J_{i,j}$. The release of server job $S_{i,j}$ is governed by the following rule.

> **Release Rule.** $S_{1,j}$ is released when $J_{1,j}$ is released. For $i \neq 1$, $S_{i,j}$ is released after every $S_{k,j}$ completes where $\tau_k \in pred(\tau_i)$.

The exact release time of a server job $S_{i,j}$ corresponding to a non-source node τ_i depends on the corresponding server scheduling policy. However, $J_{i,j}$ is released as soon as the j^{th} jobs of all predecessor nodes of τ_i complete. This allows us to (potentially) execute $J_{i,j}$ on the slack (if any) produced by the jobs of predecessor nodes (as discussed in Sec. IV-A).

Def. 2. *A server job* $S_{i,j}$ depends *on server job* $S_{k,j}$ *if* $\tau_k \in anc(\tau_i)$ *holds.*

We do not require any particular scheduling policy to be used to schedule the per-node reservations, only that the scheduler gives bounded response times and allows for the preemption of DAG nodes. However, for the sake of illustration, we henceforth assume server jobs are scheduled by a global-earliest-deadline-first (G-EDF) scheduler. The budget of a server job is managed by the following rules.

> **Replenishment Rule.** The budget of $S_{i,j}$ is replenished to C_i when it is released.
>
> **Consumption Rule.** $S_{i,j}$ consumes one unit of budget per unit of time when it is scheduled.

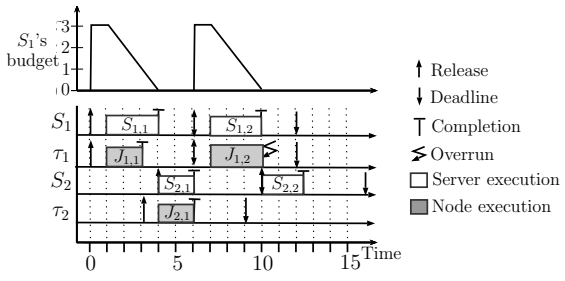


Fig. 3: Illustration of a schedule with reservations.

A server job *completes* when its budget is exhausted.

A job $J_{i,j}$ executes on its reservation $S_{i,j}$. If $J_{i,j}$ completes execution before $S_{i,j}$'s budget is exhausted, then $J_{i,j}$ *underruns* its budget. In contrast, $J_{i,j}$ *overruns* its budget if it does not complete execution at or before $S_{i,j}$'s budget is exhausted.

Ex. 1 (Cont'd)**.** *Fig. 3 shows a schedule of τ_1 and τ_2 from the DAG G in Fig. 1(a). At time 0, $J_{1,1}$ is released. Assume that $C_1 = 3$. Then, at time 0, the server $S_{1,1}$ is also released and its budget is replenished at 3.0 units. $S_{1,1}$ is scheduled at time 1 and consumes budget until time 4 when its budget is exhausted. $J_{1,1}$ starts execution at time 1 when $S_{1,1}$ begins and completes at time 3. Thus, $J_{1,1}$ underruns its budget. Since τ_2's only predecessor is τ_1, $J_{2,1}$ is released at time 3. $S_{2,1}$ is released at time 4 when $S_{1,1}$ completes. Job $J_{1,2}$ overruns as it does not complete by time 10 when $S_{1,2}$ exhausts its budget.*

B. Dealing with Budget Overruns

A job $J_{i,j}$ can no longer execute on its reservation after it overruns. Thus, any job $J_{k,\ell}$ that depends on $J_{i,j}$ is prevented from executing even though its reservation is scheduled. To allow $J_{i,j}$ to execute on $J_{k,\ell}$'s reservation, we give rules below to allocate jobs to servers.

Def. 3. *Job $J_{i,j}$ is* complete *(resp.,* incomplete*) at time t if it has (resp., has not) completed execution at or before t. Job $J_{1,j}$ of source task τ_1 is* ready *(resp.,* not ready*) at time t if it is incomplete and $J_{1,j}$ is released at or before (resp., after) time t. Job $J_{i,j}$ of a non-source task τ_i is* ready *(resp.,* not ready*) at time t if it is incomplete and each job (resp., at least one job) $J_{k,\ell}$ on which $J_{i,j}$ depends is complete (resp., incomplete) at time t.*

Job allocation rules. Let $S_{i,j}$ be a server job scheduled at time t and $\mathcal{J}(t)$ be the set of overrunning ready jobs at time t. Jobs are executed on $S_{i,j}$ via the following rules.

- **B1** If $J_{i,j}$ is ready at time t, then $J_{i,j}$ is scheduled on $S_{i,j}$.
- **B2** If $J_{i,j}$ is not ready at t, then, by Def. 3, there is at least one incomplete job $J_{k,\ell} \in dep(J_{i,j})$ at t. Let $\mathcal{J}_{i,j}(t)$ be the set of ready jobs $J_{k,\ell}$ in $dep(J_{i,j})$ at t.
  - **B2.1** If an unscheduled job in $\mathcal{J}_{i,j}(t)$ exists at time t, then select such a job $J_{k,\ell} \in \mathcal{J}_{i,j}(t)$ to execute on $S_{i,j}$.
  - **B2.2** Else, if an unscheduled job in $\mathcal{J}(t)$ exists at time t, then select such a job $J_{k,\ell} \in \mathcal{J}(t)$ to execute on $S_{i,j}$.

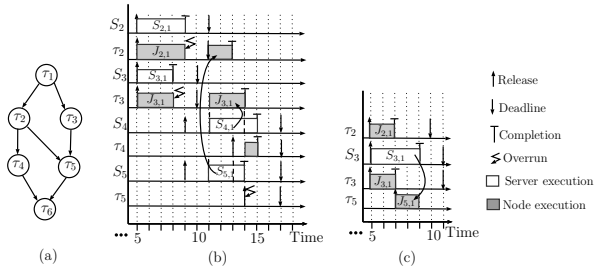


Fig. 4: (a) A DAG G, (b) a schedule of G with node overruns, and (c) a schedule of G with slack reallocation.

Concurrency. Rules B2.1 and B2.2 can be applied simultaneously at time t when multiple server jobs are scheduled concurrently. If the rules are not applied atomically, then different server jobs may select the same job to schedule. This can occur when a server job selects some job $J_{k,\ell}$ using Rule B2.1, and before $J_{k,\ell}$ is scheduled, another server job also selects $J_{k,\ell}$. We can prevent this race condition by marking a job as scheduled as soon as it is selected by Rules B2.1 and B2.2 with a single atomic operation. In a practical implementation, this can be done through a mutual exclusion lock, or through an operation such as test-and-set.

Ex. 2. *Fig. 4(a) shows a DAG G and Fig. 4(b) shows a schedule of tasks τ_2, τ_3, τ_4, and τ_5 of G. Assume that $J_{1,1}$ completes execution at or before time 5. At time 5, server jobs $S_{2,1}$ and $S_{3,1}$ are scheduled. By Rule B1, $J_{2,1}$ and $J_{3,1}$ are scheduled on $S_{2,1}$ and $S_{3,1}$, respectively, at time 5. Jobs $J_{2,1}$ and $J_{3,1}$ overrun their budgets at times 8 and 9, respectively. At time 9, $S_{4,1}$ and $S_{5,1}$ are released, and they are scheduled at time 11. Both $J_{4,1}$ and $J_{5,1}$ are not ready at time 11. $J_{2,1}$ and $J_{3,1}$ form $\mathcal{J}_{5,1}(11)$ and only $J_{2,1}$ form $\mathcal{J}_{4,1}(11)$ at time 11. Assume that $J_{2,1}$ is first selected to execute on $S_{5,1}$ at time 11 by Rule B2.1. Because $J_{2,1}$ is marked as scheduled as soon as it is selected by $S_{5,1}$, there are no remaining unscheduled jobs in $\mathcal{J}_{4,1}(11)$. $S_{4,1}$ then executes the remaining overrunning job $J_{3,1}$ by Rule B2.2.*

Based on Rules B1–B2.2, we have the following lemmas, which we use in Sec. IV.

Lemma 1. *If $S_{i,j}$ is scheduled at time t, but $J_{i,j}$ is not ready at time t, then a job $J_{k,j} \in dep(J_{i,j})$ is ready at time t.*

Proof. We first show that $i \neq 1$. Assume $i = 1$. By the Release Rule, $S_{1,j}$ is not scheduled before $J_{1,j}$'s release. By Def. 3, $J_{1,j}$ is ready at or after its release until its completion. Thus, $J_{1,j}$ is either ready or complete at time t, implying $i \neq 1$.

Now, since $J_{i,j}$ is not ready at time t and $i \neq 1$, by Def. 3, a job in $dep(J_{i,j})$ is incomplete at time t. Let $J_{k,j} \in dep(J_{i,j})$ be an incomplete job with the maximum number of nodes in the path from τ_k to τ_i. If $k = 1$ holds, then by the Release Rule, both $S_{k,j}$ and $J_{k,j}$ are released before time t. Thus, by Def. 3, $J_{k,j}$ is ready at time t. Otherwise, if $k > 1$, then since the number of nodes in the path from τ_k to τ_i is maximal, any job $J_{\ell,j}$ with $\tau_\ell \in anc(\tau_k)$ is complete at time t. Thus, by Def. 3, $J_{k,j}$ is ready at time t. □

Lemma 2. *If $S_{i,j}$ is scheduled at time t, but $J_{i,j}$ is not ready at time t, then a job $J_{k,j} \in dep(J_{i,j})$ is scheduled at time t.*

Proof. By Lem. 1, there is a job $J_{k,j} \in dep(J_{i,j})$ that is ready at time t. By Rules B2.1 and B2.2, some such job $J_{k,j}$ is scheduled on either $S_{i,j}$ or some other server job. □

Progress and job selection policies. Using Rule B2.1, jobs in $\mathcal{J}_{i,j}(t)$ (upon which $J_{i,j}$ depends) are deliberately given priority to execute on $S_{i,j}$ over jobs in $\mathcal{J}(t) \setminus \mathcal{J}_{i,j}(t)$. This ensures that a job $J_{i,j}$ waiting on jobs in $dep(J_{i,j})$ is making progress towards its completion. Also, as discussed later in Sec. IV, to analytically account for the worst-case scenario, $J_{i,j}$ must pessimistically assume that only its reservation $S_{i,j}$ can allow $J_{i,j}$ to make progress. As a result, the method to select incomplete jobs from $\mathcal{J}_{i,j}(t)$ and $\mathcal{J}(t)$ in Rules B2.1 and B2.2, respectively, only affects average-case performance and does not impact analytical DAG failure-rate bounds.

C. Slack Reallocation

By the job allocation rules above, a server job $S_{i,j}$ does not execute any job after $J_{i,j}$ completes execution, *i.e.*, the slack produced by an underrunning job is not used. Reclaiming such slack helps to avoid overruns by other jobs. We now propose slack-reallocation policies to utilize unused slack.

To take advantage of slack reallocation in our drop-rate analysis, we prioritize jobs of successor nodes when an underruning job's slack is reallocated. For each node τ_i, we select a node $\tau_k \in succ(\tau_i)$ as a *preferred successor*, denoted as $pref(\tau_i)$. $pref(\tau_i)$ is determined offline, as discussed later. We define a *preferred successor chain* of τ_i as follows.

Def. 4. *(Preferred Successor Chain.) The* preferred successor chain $pchain(\tau_i)$ *of τ_i is a sequence of tasks $(v_1, v_2, \cdots, v_k)$ such that $v_1 = \tau_i$ and for any $1 \leq j < k$, $v_{j+1} = pref(v_j)$.*

Slack reallocation rules. Let $S_{i,j}$ be a server job scheduled at time t and suppose $J_{i,j}$ is complete at time t. Let $\mathcal{J}(t)$ denote the set of overrunning jobs that are not complete at time t and $\mathcal{J}_r(t)$ be the set of ready jobs at time t. Jobs are executed on $S_{i,j}$ according to the following rules.

- **H1** If there is a ready but not scheduled job $J_{k,\ell}$ such that $\tau_k \in pchain(\tau_i)$, then execute $J_{k,\ell}$ in $S_{i,j}$.
- **H2** Otherwise, select a job (if any) via the following rules.
  - **H2.1** If an unscheduled job in $\mathcal{J}(t)$ exists at time t, then select such a job $J_{k,\ell} \in \mathcal{J}(t)$ to execute on $S_{i,j}$.
  - **H2.2** Else, if an unscheduled job in $\mathcal{J}_r(t)$ exists at time t, then select such a job $J_{k,\ell} \in \mathcal{J}_r(t)$ to execute on $S_{i,j}$.

Similar to Rules B2.1 and B2.2, when a job is selected by rules H2.1 and H2.2, it is immediately marked as scheduled to prevent a possible race condition.

Ex. 2 (Cont'd)**.** *Fig. 4(c) shows a schedule of τ_2, τ_3, and τ_5 of G in Fig. 4(a). At time 7, $J_{3,1}$ completes execution and produces 2.0 units of slack. Since both $J_{2,1}$ and $J_{3,1}$ are complete at time 7, $J_{4,1}$ and $J_{5,1}$ are released at time 7. Assume that τ_5 and τ_6 are preferred successors of τ_3 and*

Algorithm 1 Procedure for assigning preferred successors.

Variables:
$\mathcal{O}$: An indexing policy

1: **procedure** PREFSUCC(G, $\mathcal{O}$)
2: Index nodes of G according to policy $\mathcal{O}$
3: **for** $i \in \{1, 2, \cdots, n\}$ **do**
4: **for** $\tau_j \in pred(\tau_i)$ **do**
5: **if** $pref(\tau_j) =$ NULL **then**
6: $pref(\tau_j) := \tau_i$

τ_5, *respectively. Therefore,* $pchain(\tau_3) = (\tau_5, \tau_6)$. *Since* $J_{5,1}$ *is ready at time 7, it is executed on* $S_{3,1}$ *at time 7 by Rule H1.*

Determining preferred successors. Unlike the job selection policy of Rules B2.1 and B2.2, the method of determining a node's preferred successor allows us to guarantee the progress of certain jobs in the system, thereby improving a DAG's analytical drop rate. To set each node's preferred successor, we use the method PREFSUCC in Alg. 1. PREFSUCC first indexes the nodes according to a specified policy (line 2). We consider three node-indexing policies: **(i)** random, **(ii)** minimum indegree first, and **(iii)** maximum outdegree first. Intuitively, minimum degree first attempts to minimize the number of nodes sharing a preferred successor, better balancing the distribution of slack. In contrast, maximum outdegree first prioritizes nodes in the congested parts of a DAG. After indexing, PREFSUCC iteratively considers a task τ_i and assigns τ_i as the preferred successor of all its predecessor nodes that do not have any preferred successor yet (lines 3–6).

D. Aborting a DAG Invocation

Even with budget- and slack-reallocation, DAG invocations may need to be aborted to maintain schedulability. We abort a DAG invocation if its sink node's reservation budget is exhausted but the DAG invocation has incomplete jobs.

Let R be a response-time bound of G under any scheduling algorithm $\mathcal{A}$. By the Release Rule, when scheduled by $\mathcal{A}$, the difference between $S_{n,j}$'s completion and $S_{1,j}$'s release is at most R time units. Thus, response times of non-aborted invocations of G are entirely dependent on the response-time bound of the reservation scheduler.

IV. DROP-RATE ANALYSIS

In this section, we analyze the probability that any invocation of a DAG of interest G is dropped. Note that Rules B1–B2.2 and H1–H2.2 only pertain to a single DAG, and by assumption, the employed scheduler ensures bounded DAG response times at the server level. Thus, a given DAG's drop rate does not depend on the scheduling, budgeting, or drop rate of any other DAG. In our analysis, we assume the following.

A1. The per-node pWCETs e_k in G are mutually independent.

DAG abort condition. As stated previously, when some job $J_{n,j}$ of the sink node τ_n overruns its budget, the j^{th} DAG invocation is aborted. Unfortunately, the exact conditions under which $J_{n,j}$ overruns are complicated by the fact that overrunning jobs can delay the execution of other jobs.

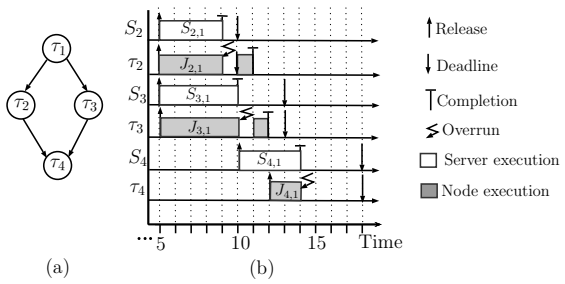


Fig. 5: (a) A DAG G and (b) a schedule of G.

Ex. 3. *Consider the DAG* G *in Fig. 5, which executes on* $m = 2$ *processors. At times 9 and 10,* $J_{2,1}$ *and* $J_{3,1}$ *overrun the budget of* $S_{2,1}$ *and* $S_{3,1}$ *by 1.0 time unit, respectively. Due to Rule B2.1,* $S_{4,1}$ *schedules* $J_{2,1}$ *at time 10 and executes it to completion. Then* $S_{4,1}$ *schedules* $J_{3,1}$ *at time 11, which also executes to completion before* $J_{4,1}$ *becomes ready and begins execution in* $S_{4,1}$ *at time 12.*

In Ex. 3, despite the execution time of $J_{4,1}$, $e_4 = 3$, being smaller than the server budget, $C_4 = 4$, $J_{4,1}$ cannot complete by time 14 when $S_{4,1}$ exhausts its budget, triggering a DAG abort. This is because when $S_{4,1}$ is first scheduled, $J_{4,1}$ and two jobs on which $J_{4,1}$ depends, $J_{2,1}$ and $J_{3,1}$, have a total remaining execution time of 5.0 units. This value, which we call the *demand* of τ_4, exceeds C_4, so $J_{4,1}$ overruns its budget.

Def. 5. *(Demand.) The* demand *of a node* τ_k *is represented by the nonnegative RV* β_k. *On DAG* G*'s* j^{th} *invocation,* β_k *equals the total remaining execution time of all jobs in* $dep(J_{k,j}) \cup \{J_{k,j}\}$ *when* $S_{k,j}$ *is first scheduled.*

As seen in Ex. 3, when $\beta_n > C_n$ holds for the sink node τ_n, $S_{n,j}$ does not have sufficient budget to complete all jobs in $dep(J_{n,j}) \cup \{J_{n,j}\}$. Thus, $J_{n,j}$ overruns only when $\beta_n > C_n$ holds, making $P(\beta_n > C_n)$ an upper bound on the probability of aborting an invocation of G.

Upper-bounding demand. Unfortunately, it may be difficult to obtain the exact demand of a node. Thus, for each node τ_k, we want to find a RV β_k^+ such that $\beta_k^+ \geq_0 \beta_k$. Recall that $\beta_k^+ \geq_0 \beta_k$ implies $\beta_k^+ \geq_1 \beta_k$, thus computing $P(\beta_n^+ > C_n)$ will allow us to upper-bound $P(\beta_n > C_n)$, or the probability of aborting G.

Base case. We begin finding the PDF of β_n^+ by analyzing the PDF of β_1^+ for the source node τ_1. Unlike other nodes in G, the source node has no predecessors. Therefore $dep(J_{1,j}) \cup \{J_{1,j}\} = \{J_{1,j}\}$. By Def. 5, we obtain $\beta_1 = e_1$. This leads to

Observation 1. *For the source node* τ_1, $\beta_1^+ = e_1$ *satisfies* $\beta_1^+ \geq_0 \beta_1$.

Using Obs. 1 as the base case, we proceed to compute β_k^+ for each node τ_k by structurally inducting through the graph. First, in Sec. IV-A, we consider the effect of slack reallocation on β_k^+. Second, in Sec. IV-B, we consider the effect of overrun management on β_k^+. Third, in Sec. IV-C, we combine the effect of slack reallocation and overrun management and give

a method to compute an upper bound on the DAG drop rate. Finally, in Sec. IV-D, we give a more practical alternative method for computing such an upper bound.

A. Accounting for Slack Reallocation

In this subsection, we examine how slack reallocation affects the relation between the demand of a node τ_k and the demand of its predecessor nodes. To quantify this relationship, we first define the "slack" of a node.

Def. 6. *(Slack.) Node τ_i's* slack *is given by the nonnegative RV q_i. On any j^{th} invocation of DAG G, the observed value of q_i is equal to the remaining budget of $S_{i,j}$ when $J_{i,j}$ completes.*

Intuitively, the slack of a node τ_i indicates "how much" jobs of τ_i underrun its budget. Thus, computing a lower bound of q_i allows us to upper-bound how much slack reallocation can lower the demand of successor nodes of τ_i. In the following lemma, we lower-bound q_i by a function of τ_i's demand β_i^+.

Lemma 3. *For each node τ_i, $q_i \geq_0 \max(0, C_i - \beta_i^+)$.*

Proof. Let t_0 be the time when a server job $S_{i,j}$ is first scheduled, and let t_1 be the later of t_0 and the time when $J_{i,j}$ completes. According to Def. 6, this lemma states that at t_1, the remaining budget of $S_{i,j}$ is at least $\max(0, C_i - \beta_i^+)$. Assume to the contrary that **(A)** this remaining budget is less than $\max(0, C_i - \beta_i^+)$ at t_1. We consider two cases.

Case 1. $\beta_i^+ > C_i$. Asm. (A) implies that at t_1, $S_{i,j}$'s remaining budget is less than $\max(0, C_i - \beta_i^+) = 0$. However, budgets are always non-negative. Contradiction.

Case 2. $\beta_i^+ \leq C_i$. By Asm. (A), at t_1, $S_{i,j}$'s remaining budget is less than $C_i - \beta_i^+ \geq 0$. According to the Replenishment Rule, the remaining budget of $S_{i,j}$ is C_i at t_0. Thus, by the Consumption Rule, $S_{i,j}$ is scheduled for more than β_i^+ time units between t_0 and t_1. By Def. 5, the total remaining execution time of jobs in $dep(J_{i,j}) \cup \{J_{i,j}\}$ at t_0 is β_i, which is state-wise at most β_i^+. Thus, there must exist some time $t \in [t_0, t_1)$ when $S_{i,j}$ is scheduled, but a job in $dep(J_{i,j}) \cup \{J_{i,j}\}$ is not. However, if $J_{i,j}$ is not ready at t, some job in $dep(J_{i,j})$ must be scheduled due to Lem. 2, and if $J_{i,j}$ is ready at t, then $J_{i,j}$ must be scheduled due to Rule B1. Contradiction. □

When a node τ_i has positive slack, some of this slack can be used to schedule jobs of its preferred successor node τ_k via Rule H1. Consider the example below, illustrated by Fig. 6.

Ex. 4. *Nodes of a DAG G are scheduled on two processors. In G, τ_5 is assigned as the preferred successor of both τ_2 and τ_3 by* PREFSUCC. *$J_{2,1}$ (resp., $J_{3,1}$) has 2.0 (resp., 1.0) units of slack. At time 10, both $J_{2,1}$ and $J_{3,1}$ are complete. Thus, $J_{5,1}$ is ready at time 10. Since $\tau_5 = pref(\tau_2) = pref(\tau_3)$, $S_{3,1}$ uses Rule H1 to schedule $J_{5,1}$ from time 10 to time 11.*

In Ex. 4, we see that despite $q_2 = 2$, due to the later completion time of $J_{3,1}$, $J_{5,1}$ is only able to be scheduled by Rule H1 for the minimum of the two slacks q_2 and q_3. Thus, $\beta_5 = e_5 - q_3 = 3 - 1 = 2$. In contrast, $J_{4,1}$, whose node τ_4 is not the preferred successor of τ_2 and τ_3, cannot be scheduled

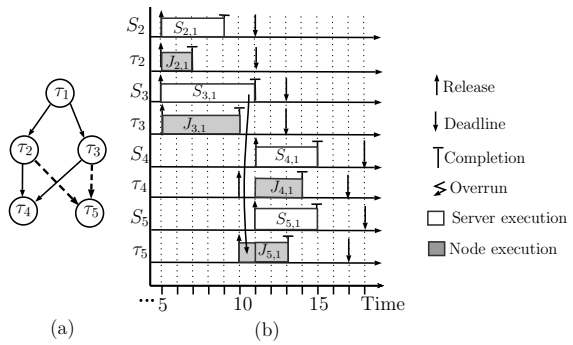


Fig. 6: (a) A DAG G (dashed edges indicate preferred successors) and (b) a schedule of G.

by Rule H2.1 due to the presence of $J_{5,1}$. From this example, we see that only the preferred successor is guaranteed to be scheduled during the slack of a node. Furthermore, because our method is scheduler agnostic, we must account for the worst case where only the smallest slack of the predecessor nodes of a preferred successor node τ_k may be *usable* in decreasing its demand. This discussion leads us to the following definition.

Def. 7. *(Usable Slack.) RV Ψ_k denotes the* usable slack *of τ_k. If τ_k is the preferred successor of each $\tau_i \in pred(\tau_k)$, then Ψ_k=$\min(\{\max(0, C_i - \beta_i^+) | \tau_i \in pred(\tau_k)\})$. Otherwise, Ψ_k=0.*

Using Def. 7, we proceed to upper-bound β_k using a function of β_i^+ of each predecessor node τ_i of τ_k.

Lemma 4. *If $\Psi_k > 0$, then $\beta_k \leq_0 e_k - \min(e_k, \Psi_k)$.*

Proof. Suppose job $J_{\ell,j}$ completed the latest among $pred(J_{k,j})$. Let t_0 be the time $J_{\ell,j}$ completes, and let t_1 be the time $S_{\ell,j}$ completes. Since $J_{\ell,j}$ completes at t_0, by Def. 1, all jobs in $dep(J_{k,j})$ complete by time t_0.

Claim 1. *$S_{\ell,j}$ has at least $\Psi_k > 0$ time units of remaining budget at t_0.*

Proof. By Def. 7, Ψ_k is state-wise at most $\max(0, C_\ell - \beta_\ell^+)$. Thus, given $\Psi_k > 0$, we have $\max(0, C_\ell - \beta_\ell^+) \geq_0 \Psi_k > 0$, so by Lem. 3, $q_\ell \geq_0 \Psi_k$. Hence, the claim is true by Def. 6. □

By Clm. 1, $S_{\ell,j}$ has positive budget remaining at t_0. Thus, by the definition of t_1, $t_1 \geq t_0$.

Claim 2. *$J_{k,j}$ is scheduled for at least $\min(e_k, \Psi_k)$ time units in $[t_0, t_1)$.*

Proof. All jobs in $dep(J_{k,j})$ are complete at t_0, so $J_{k,j}$ is ready during $[t_0, t_1)$ unless it completes. By Clm. 1 and the Consumption Rule, $S_{\ell,j}$ must be scheduled for at least Ψ_k time units during $[t_0, t_1)$ to exhaust its budget at t_1, so by Rule H1, $J_{k,j}$ is scheduled for at least $\min(e_k, \Psi_k)$ time units during $[t_0, t_1)$. □

Using the above two claims, all jobs in $dep(J_{k,j})$ are complete, and $J_{k,j}$ has already executed for at least $\min(e_k, \Psi_k)$ time units by t_1. Because of the Release Rule, $S_{k,j}$ can only be scheduled after t_1, resulting in the total remaining

execution time of jobs in $dep(J_{k,j}) \cup \{J_{k,j}\}$ being at most $e_k - \min(e_k, \Psi_k)$. By Def. 5, $\beta_k \leq_0 e_k - \min(e_k, \Psi_k)$ □

By Lem. 4, $\beta_k^+ = e_k - min(e_k, \Psi_k)$ is a valid assignment for β_k^+ that satisfies $\beta_k^+ \geq_0 \beta_k$. This assignment establishes a relation between β_k^+ and β_i^+ for $\tau_i \in pred(\tau_k)$ that accounts for slack reallocation.

B. Accounting for Overruns

In this subsection, we examine how job overruns affect the relation between the demand of a node τ_k and the demand of the predecessor nodes of τ_k. To quantify this relationship, we first define the *overrun* of a node.

Def. 8. *(Overrun.) The* overrun *of a node τ_i is represented by the nonnegative RV o_i. On any j^{th} invocation of DAG G, the observed value of o_i equals the total remaining execution time of all jobs in $dep(J_{i,j}) \cup \{J_{i,j}\}$ when $S_{i,j}$ exhausts its budget.*

State-wise upper-bounding the overrun of τ_i allows us to place an upper bound on the demand of successor nodes of τ_i. The next lemma upper-bounds o_i by a function of β_i^+.

Lemma 5. *For each node τ_i, $o_i \leq_0 \max(0, \beta_i^+ - C_i)$.*

Proof. Suppose $S_{i,j}$ is first scheduled at t_0 and exhausts its budget at t_1. According to Def. 8, the lemma states that the total remaining execution time of all jobs in $dep(J_{i,j}) \cup \{J_{i,j}\}$ at t_1 is at most $\max(0, \beta_i^+ - C_i)$. Assume to the contrary that **(B)** their total remaining execution time is more than $\max(0, \beta_i^+ - C_i)$ at t_1. We first prove the following.

Claim 3. *If a job in $dep(J_{i,j}) \cup \{J_{i,j}\}$ is incomplete at time t_1, then the jobs in $dep(J_{i,j}) \cup \{J_{i,j}\}$ are scheduled for at least C_i time units during $[t_0, t_1)$ by the system.*

Proof. By Def. 3, $J_{i,j}$ is ready only after all jobs in $dep(J_{i,j})$ are complete (note that $dep(J_{i,j}) = \emptyset$ if $i = 1$). Hence, by Lem. 2 and Rule B1, jobs in $dep(J_{i,j}) \cup \{J_{i,j}\}$ are scheduled whenever $S_{i,j}$ is scheduled. Thus, these jobs execute for at least C_i time units during $[t_0, t_1)$. □

We now prove the lemma. By Asm. (B), there are incomplete jobs in $dep(J_{i,j}) \cup \{J_{i,j}\}$ at time t_1. By Clm. 3, jobs in $dep(J_{i,j}) \cup \{J_{i,j}\}$ execute for at least C_i time units during $[t_0, t_1)$. Thus, by Def. 5, their total remaining execution time at time t_1 is at most $\beta_i - C_i$, which is state-wise at most $\max(0, \beta_i^+ - C_i)$ at time t_1, contradicting Asm. (B). □

In Ex. 3, we see that $\beta_4 = o_2 + o_3 + e_4$. Generalizing from this example, in the worst case, the *total overrun* of all predecessor nodes of τ_k increases the demand of τ_k.

Def. 9. *(Total Overrun). The* total overrun *of all predecessor nodes of τ_k, denoted by the nonnegative RV Φ_k, is given by $\Phi_k = \sum_{\tau_i \in pred(\tau_k)} \max(0, \beta_i^+ - C_i)$.*

Using the notion of total overrun, we now upper-bound β_k as a function of β_i^+ for each predecessor node τ_i of τ_k.

Lemma 6. *For each node τ_k, $\beta_k \leq_0 e_k + \Phi_k$.*

Proof. By Def. 8, for each job $J_{i,j} \in dep(J_{k,j})$, the remaining execution time of $J_{i,j}$ and the jobs on which $J_{i,j}$ depends when $J_{i,j}$'s server job $S_{i,j}$ exhausts its budget is o_i. Since the Release Rule states that $S_{k,j}$ can begin execution when each such $S_{i,j}$ exhausts its budget, the total remaining execution of jobs in $dep(J_{k,j})$ when $S_{k,j}$ is first scheduled is at most $\sum_{\tau_i \in pred(\tau_k)} o_i$. By Lem. 5, each o_i term is state-wise at most $\max(0, \beta_i^+ - C_i)$, so this total remaining execution is at most Φ_k by Def. 9. Because e_k is the execution time of $J_{k,j}$, when $S_{k,j}$ is first scheduled, by Def. 5, we have $\beta_k \leq_0 e_k + \Phi_k$. □

With Lem. 6, when jobs overrun their budget, $\beta_k^+ = e_k + \Phi_k$ becomes a valid assignment for β_k^+ that satisfies $\beta_k^+ \geq_0 \beta_k$. This assignment establishes a relation between β_k^+ and β_i^+ for $\tau_i \in pred(\tau_k)$ that accounts for budget overruns.

C. Computing the Drop Rate

In the previous two subsections, we established a relation between β_k^+ and β_i^+ for each $\tau_i \in pred(\tau_k)$ that accounts for slack reallocation and overruns, respectively. In this subsection, we unify the results from the previous subsections and compute an upper bound on the drop rate of DAG G. To aid in this, we first define the RV Δ_k for each node τ_k as follows.

$$\Delta_k = \begin{cases} -\Psi_k & \text{if } \Psi_k > 0 \\ \Phi_k & \text{otherwise} \end{cases} \tag{1}$$

We then use Δ_k to state-wise bound the value of β_k.

Theorem 1. *For each node τ_k, $\beta_k \leq_0 \max(0, \Delta_k + e_k)$.*

Proof. We examine the two cases of (1).

Case 1. $\Psi_k > 0$. By Lem. 4, $\beta_k \leq_0 e_k - \min(\Psi_k, e_k) = \max(e_k - e_k, e_k - \Psi_k) = \max(0, -\Psi_k + e_k) = \max(0, \Delta_k + e_k)$.

Case 2. $\Psi_k \leq 0$. By Lem. 6, we have $\beta_k \leq_0 e_k + \Phi_k \leq_0 \max(0, e_k + \Phi_k) = \max(0, \Delta_k + e_k)$. □

Drop rate computation. Thm. 1 gives us a state-wise upper bound on β_k for each node τ_k. Therefore, $\max(0, \Delta_k + e_k)$ is a valid assignment of β_k^+ for each node τ_k that satisfies $\beta_k^+ \geq_0 \beta_k$. Using this assignment, each β_k^+ can be expanded into an expression of e_k and Δ_k. Additionally, using (1) and Defs. 7 and 9, Δ_k can be written as an expression of β_i^+ for each $\tau_i \in pred(\tau_k)$. Thus, by expanding out the expressions of β_i^+ for each $\tau_i \in anc(\tau_k)$ in a similar fashion, we have the following observation.

Observation 2. *Let $E_k = \{e_i \mid \tau_i \in anc(\tau_k)\} \cup e_k$. β_k^+ for each τ_k can be expressed as a function of E_k. We denote that function as $\beta_k^+ = \mathcal{F}_k(E_k)$.*

To compute an upper bound on the drop rate of G, $P(\beta_n^+ > C_n)$, we can then substitute β_n^+ with $\mathcal{F}_n(E_n)$. To avoid the difficult process of computing the PDF of a function of multiple RVs, we observe that $\mathcal{F}_n(E_n)$ is equal to $\mathcal{F}_n(\{x_1, x_2, ..., x_n\})$ when $e_i = x_i$ for each $e_i \in E_n$. In other words, by conditioning $P(\beta_n^+ > C_n)$ on e_i for each node τ_i, we have the following.

Observation 3.

$$P(\beta_n^+ > C_n \mid \forall i : e_i = x_i) = P(\mathcal{F}_n(\{x_1, x_2, ...\}) > C_n) \tag{2}$$

Now, applying to Obs. 3 the law of total probability, which states that $P(A) = \sum_{i=-\infty}^{\infty} P(A \mid X = i)P(X = i)$ for some event A and a RV X, and Asm. A1, we have the following.

Theorem 2. *Let $\mathbb{I}_i$ be the set containing all values of e_i where $P(e_i = x_i) \neq 0$. The drop rate of G, $P(\beta_n^+ > C_n)$, is*

$$\sum_{x_1 \in \mathbb{I}_1, x_2 \in \mathbb{I}_2, ...} \left[P(\mathcal{F}_n(\{x_1, x_2, ...\}) > C_n) \prod_{i=1}^{n} P(e_i = x_i) \right]. \tag{3}$$

Runtime issues. Unfortunately, computing $P(\beta_n^+ > C_n)$ using (3) is computationally expensive. This is because there are $\prod_{i=1}^{n} |\mathbb{I}_i|$ terms in the summation, and each term requires the multiplication of n more terms. This results in a time complexity of $O(max(|\mathbb{I}_i| : i = 1...n)^{n+1})$ where n is the number of nodes in G. We therefore explore an alternative method to compute $P(\beta_n^+ > C_n)$.

D. Alternate Drop Rate Computation

In this subsection, we present an alternative drop rate computation method. $\mathcal{F}_n$ is obtained by iteratively expanding β_k^+ for each τ_k into $\max(0, \Delta_k + e_k)$, and then using (1) and Defs. 7 and 9 to expand each Δ_k. This process to obtain $\mathcal{F}_n$ requires applying the following primitive operations: **(i)** adding a constant to a RV; **(ii)** negating a RV; **(iii)** adding two RVs; **(iv)** determining the maximum between 0 and a RV; **(v)** determining the minimum of multiple RVs; and **(vi)** computing Δ_k from Φ_k and Ψ_k. In the following Props. 1 to 5, we show how the above operations transform the PDFs of their input RVs into the PDFs of their output RVs. After applying the transforms accordingly, we can obtain the PDF of β_n^+.

Props. 1 and 2 are self-evident, Props. 3 is a well-known convolution property, and we give proofs for Props. 4 and 5

Property 1. *For a RV X and a constant k, the PDF of $X + k$ is given by $f_{X+k}(x) = f_X(x - k)$.*

Property 2. *For a RV X, the PDF of $-X$ is given by $f_{-X}(x) = f_X(-x)$.*

Property 3. *For two independent RVs X and Y, the PDF of $X + Y$ is given by the convolution of f_X and f_Y, where $f_{X+Y}(x) = \sum_{i=-\infty}^{\infty} f_X(i - x)f_Y(i)$.*

Property 4. *For a RV X, the PDF of $\max(0, X)$ is given by*

$$f_{\max(0,X)}(x) = \begin{cases} 0 & x < 0 \\ F_X(0) & x = 0 \\ f_X(x) & x > 0. \end{cases}$$

Proof. We examine the three possible cases of $f_{\max(0,X)}(x)$.

Case 1. $x < 0$. $\max(0, X)$ cannot be negative. Therefore, $P(\max(0, X) = x) = 0$.

Case 2. $x = 0$. This occurs when $X \leq 0$. Thus, $P(\max(0, X) = 0) = P(X \leq 0)$.

Case 3. $x > 0$. Here, $\max(0, X) = X$, so $P(\max(0, X) = x) = P(X = x)$. □

Claim 4. *For a set of mutually independent RVs $\mathbf{X} = \{X_1, X_2, ..., X_n\}$, the PDF of $\min(\mathbf{X})$ is given by $f_{\min(\mathbf{X})}(x) = \overline{F}_{\min(\mathbf{X})}(x - 1) - \overline{F}_{\min(\mathbf{X})}(x)$, where*

$$\overline{F}_{\min(\mathbf{X})}(x) = \prod_{X_i \in \mathbf{X}} \overline{F}_{X_i}(x). \tag{4}$$

Proof. Consider the event where $\min(\mathbf{X}) > x$ for some value of x. This occurs when each $X_i \in \mathbf{X}$ is greater than x. Thus, we have $P(\min(\mathbf{X}) > x) = P\left(\bigwedge_{X_i \in \mathbf{X}} X_i > x\right)$. Because the RVs in $\mathbf{X}$ are independent, $P\left(\bigwedge_{X_i \in \mathbf{X}} X_i > x\right) = \prod_{X_i \in \mathbf{X}} P(X_i > x) = \prod_{X_i \in \mathbf{X}} \overline{F}_{X_i}(x)$. □

To derive how (1) transforms the PDFs of Ψ_k and Φ_k to Δ_k, we first prove the following.

Lemma 7. *For each node τ_k, the events $\Psi_k > 0$ and $\Phi_k > 0$ are mutually exclusive.*

Proof. If $\Phi_k > 0$, then by Def. 9, $\beta_i^+ - C_i > 0$ holds for some $\tau_i \in pred(\tau_k)$. Hence, $C_i - \beta_i^+ < 0$ holds, and by Def. 7, $\Psi_k = 0$. On the other hand, if $\Psi_k > 0$, then by Def. 7, for all $\tau_i \in pred(\tau_k)$, $C_i - \beta_i^+ > 0$ holds, which implies $\beta_i^+ - C_i < 0$ holds. Thus, by Def. 9, $\Phi_k = 0$. □

Property 5. *The PDF of Δ_k for each node τ_k is given by*

$$f_{\Delta_k}(x) = \begin{cases} f_{\Psi_k}(-x) & x < 0 \\ 1 - \overline{F}_{\Psi_k}(0) - \overline{F}_{\Phi_k}(0) & x = 0 \\ f_{\Phi_k}(x) & x > 0. \end{cases} \tag{5}$$

Proof. We consider the three cases in (5) separately.

Case 1. $x < 0$. Note that, by Def. 9, $\Phi_k \geq 0$. Also, from (1), either $\Delta_k = \Phi_k$ or $\Delta_k = -\Psi_k$. Because $\Phi_k \geq 0$, Case 1 can only occur when $\Delta_k = -\Psi_k$. Therefore, (5) holds by Clm. 2.

Case 2. $x = 0$. By Defs. 7 and 9, both Φ_k and Ψ_k are non-negative. By (1), $x = 0$ only occurs when $\Phi_k = 0$ and $\Psi_k = 0$. Therefore, $f_{\Delta_k}(0) = P(\Phi_k = 0, \Psi_k = 0) = 1 - P(\Psi_k > 0 \text{ or } \Phi_k > 0)$. By Lem. 7, events $\Psi_k > 0$ and $\Phi_k > 0$ are mutually exclusive. Thus, we have $P(\Psi_k > 0 \text{ or } \Phi_k > 0) = P(\Psi_k > 0) + P(\Phi_k > 0) = \overline{F}_{\Psi_k}(0) + \overline{F}_{\Phi_k}(0)$. Therefore, $P(\Phi_k = 0, \Psi_k = 0) = 1 - (\overline{F}_{\Psi_k}(0) + \overline{F}_{\Phi_k}(0))$.

Case 3. $x > 0$. According to (1), this case occurs when $\Delta_k = \Phi_k$. Thus $f_{\Delta_k}(x) = f_{\Phi_k}(x)$. □

Starting with $\beta_1^+ = e_1$ and using Props. 1–5 to apply the PDF transformation due to each primitive operation in $\mathcal{F}_n$, we can obtain the PDF of β_n^+. Using this PDF will allow us to upper-bound the drop rate of G.

Undesired independence assumption. Unfortunately, the independence requirements of Props. 3 and 4 present an issue. Computing Φ_k (resp., Ψ_k) as per Def. 9 (resp., Def. 7) requires applying Prop. 3 (resp., Prop. 4) to compute the RV

addition (resp., the min function). This requires β_i^+ for each $\tau_i \in pred(\tau_k)$ to be mutually independent. However, for any τ_i and τ_j in $pred(\tau_k)$, β_i^+ and β_j^+ may not be independent. This is a direct consequence of Obs. 2—τ_i and τ_j can have a common ancestor τ_ℓ, in which case both β_i^+ and β_j^+ are functions of e_ℓ. Using Props. 3 and 4 to compute the PDFs of Φ_k and Ψ_k for each node τ_k may not result in the true PDFs of Φ_k and Ψ_k, which are needed to compute the correct PDF of β_n^+. Hence, we proceed to bound the PDFs of Ψ_k and Φ_k and show that such bounds can be used to upper-bound $P(\beta_n^+ > C_n)$.

Drop rate computation—upper bound. To facilitate upper-bounding $P(\beta_n^+ > C_n)$, we introduce a new RV γ_k for each node τ_k. We let $\gamma_1 = e_1$ for the source node, and proceed to iteratively define the PDF of γ_k for each node τ_k as a function of the PDFs of γ_i for $\tau_i \in pred(\tau_k)$ using bounds on the PDFs of Ψ_k and Φ_k. If we show that $\gamma_k \geq_1 \beta_k^+$, then $P(\gamma_n > C_n)$ upper-bounds $P(\beta_n^+ > C_n)$.

We prove $\gamma_k \geq_1 \beta_k^+$ through induction. For the base case, $\gamma_1 \geq_1 \beta_1^+$ is trivially true since $\gamma_1 = \beta_1^+ = e_1$. For the induction step, we assume the following induction hypothesis.

IH For non-source node τ_k, $\gamma_i \geq_1 \beta_i^+$ holds for each node $\tau_i \in pred(\tau_k)$.

For the rest of this section, we give the iterative definition of γ_k, and show that given IH, $\gamma_k \geq_1 \beta_k^+$ holds. This then completes our proof of the induction step. To aid in our proof, we first give properties regarding several operations used to iteratively define γ_k. These properties follow directly from the closure properties of first-degree stochastic ordering under non-decreasing and non-increasing functions [37].

Property 6. *For RVs X, Y and a constant k, if $X \geq_1 Y$, then $X + k \geq_1 Y + k$.*

Property 7. *For RVs X and Y, if $X \geq_1 Y$, then $\max(0, X) \geq_1 \max(0, Y)$.*

Property 8. *For RVs X and Y, if $X \geq_1 Y$, then $-X \leq_1 -Y$.*

Lower-bounding Ψ_k. To iteratively define γ_k, we must first lower-bound Ψ_k. Let the RV Ψ_k^- first-degree lower bound Ψ_k for each node τ_k. The PDF of Ψ_k^- is characterized by the following. If τ_k is *not* the preferred successor of each $\tau_i \in pred(\tau_k)$, then Ψ_k^- is a RV with PDF

$$f_{\Psi_k^-}(x) = \begin{cases} 1 & x = 0 \\ 0 & otherwise. \end{cases} \tag{6}$$

If τ_k is the preferred successor of each $\tau_i \in pred(\tau_k)$, then Ψ_k^- is a RV with PDF $f_{\Psi_k^-}(x) = \overline{F}_{\Psi_k^-}(x-1) - \overline{F}_{\Psi_k^-}(x)$, where

$$\overline{F}_{\Psi_k^-}(x) = \prod_{\tau_i \in pred(\tau_k)} \overline{F}_{\max(0, C_i - \gamma_i)}(x). \tag{7}$$

The next two lemmas prove that $\Psi_k^- \leq_1 \Psi_k$.

Lemma 8. *If τ_k is not the preferred successor of each $\tau_i \in pred(\tau_k)$, then $\Psi_k^- \leq_1 \Psi_k$.*

Proof. Since τ_k is not the preferred successor, $\Psi_k = 0$ holds by Def. 7. Therefore, $P(\Psi_k = 0) = 1$. By (6), $P(\Psi_k^- = 0) = 1$ holds as well. The lemma follows. □

To prove that $\Psi_k^- \leq_1 \Psi_k$ when τ_k is the preferred successor, we require that $P(\max(0, C_i - \beta_i^+) > x)$ and $P(\max(0, C_j - \beta_j^+) > x)$ are not negatively correlated. In other words:

Lemma 9. *For τ_i and τ_j, both in $pred(\tau_k)$, $P(\max(0, C_i - \beta_i^+) > x) P(\max(0, C_j - \beta_j^+) > x) \leq P(\max(0, C_i - \beta_i^+) > x, \max(0, C_j - \beta_j^+) > x)$.*

The proof is included in an online appendix [39]. A loose justification is that, as functions of β_i^+ (resp. β_j^+), $\max(0, C_i - \beta_i^+)$ (resp., $\max(0, C_j - \beta_j^+)$) can be expressed as functions of the pWCETs of their common ancestors. Due to Asm. A1, these pWCETs are independent, so the FKG Inequality [18] can be applied to get Lem. 9.

Lemma 10. *If τ_k is the preferred successor of each $\tau_i \in pred(\tau_k)$, then $\Psi_k^- \leq_1 \Psi_k$.*

Proof. By Def. 7, $\Psi_k = \min(\{\max(0, C_i - \beta_i^+) \mid \tau_i \in pred(\tau_k)\})$, so $\Psi_k > x$ holds only if $\max(0, C_i - \beta_i^+) > x$ holds for each $\tau_i \in pred(\tau_k)$. Thus, $P(\Psi_k > x) = P\left(\bigwedge_{\tau_i \in pred(\tau_k)} \max(0, C_i - \beta_i^+) > x\right)$.

By Lem. 9, we have

$$\prod_{\tau_i \in pred(\tau_k)} \overline{F}_{\max(0, C_i - \beta_i^+)}(x) \leq P\left(\bigwedge_{\tau_i \in pred(\tau_k)} \max(0, C_i - \beta_i^+) > x\right). \tag{8}$$

We now prove the lemma by establishing a relation between each term in the rhs of (7) and a corresponding term in the lhs of (8). By IH, we have $\gamma_i \geq_1 \beta_i^+$. By Prop. 8, we have $-\gamma_i \leq_1 -\beta_i^+$. Applying Prop. 6, we get $C_i - \gamma_i \leq_1 C_i - \beta_i^+$. Finally, by Prop. 7, we have $\max(0, C_i - \gamma_i) \leq_1 \max(0, C_i - \beta_i^+)$. Therefore, each term in the rhs of (7) is at most the corresponding term in the lhs of (8). Thus, the lhs of (7), $P(\Psi_k^- > x)$, is at most the rhs of (8). Since the rhs of (8) is equal to $P(\Psi_k > x)$, we have $\Psi_k^- \leq_1 \Psi_k$. □

Upper-bounding Φ_k. Here, we give an upper bound Φ_k by introducing a RV Φ_k^+ for each node τ_k. We will later use Φ_k to define γ_k. Letting μ_X denote the mean of RV X, the PDF of Φ_k^+ is as follows.

$$f_{\Phi_k^+}(x) = \overline{F}_{\Phi_k^+}(x-1) - \overline{F}_{\Phi_k^+}(x), \tag{9}$$

where

$$\overline{F}_{\Phi_k^+}(x) = \begin{cases} \min\left(F_{\Psi_k^-}(0), \dfrac{1}{x+1} \cdot \displaystyle\sum_{\tau_i \in pred(\tau_k)} \mu_{\max(0, \gamma_i - C_i)}\right) & x \geq 0 \\ 1 & x < 0. \end{cases} \tag{10}$$

The following lemma proves that $\Phi_k^+ \geq_1 \Phi_k$.

Lemma 11. *For each node τ_k, $\Phi_k^+ \geq_1 \Phi_k$.*

Proof. We consider three cases based on the structure of $\overline{F}_{\Phi_k^+}(x)$ in (10).

Case 1. $x \geq 0$, and $F_{\Psi_k^-}(0)$ is the smaller term in the min function. Here, $P(\Phi_k^+ > x) = P(\Psi_k^- \leq 0)$ follows from (10) (and the definitions of $\overline{F}_{\Phi_k^+}(x)$ and $F_{\Psi_k^-}(0)$). As the events $\Phi_k > 0$ and $\Psi_k > 0$ are mutually exclusive by Lem. 7, we have $\Phi_k > 0 \Rightarrow \Psi_k \leq 0$. Therefore, $P(\Phi_k > 0) \leq P(\Psi_k \leq 0)$. Thus, since $x \geq 0$, we have $P(\Psi_k \leq 0) \geq P(\Phi_k > 0) \geq P(\Phi_k > x)$. Hence, because Lems. 8 and 10 imply $P(\Psi_k^- \leq 0) \geq P(\Psi_k \leq 0)$, we have $P(\Phi_k^+ > x) = P(\Psi_k^- \leq 0) \geq P(\Psi_k \leq 0) \geq P(\Phi_k > x)$. Thus, $\Phi_k^+ \geq_1 \Phi_k$.

Case 2. $x \geq 0$ and $F_{\Psi_k^-}(0)$ is the larger term in the min function. Assume for a contradiction that for some $x \geq 0$, $P(\Phi_k^+ > x) < P(\Phi_k > x)$ holds. In this case, by the definition of $\overline{F}_{\Phi_k^+}(x)$ in (10), we have $\frac{1}{x+1} \cdot \sum_{\tau_i \in pred(\tau_k)} \mu_{\max(0,\gamma_i - C_i)} < P(\Phi_k > x)$. By IH, $\gamma_i \geq_1 \beta_i^+$ holds for each $\tau_i \in pred(\tau_k)$. Then, by Props. 6 and 7, $\max(0, \gamma_i - C_i) \geq_1 \max(0, \beta_i^+ - C_i)$. Hence, $\mu_{\max(0,\gamma_i - C_i)} \geq \mu_{\max(0,\beta_i^+ - C_i)}$. Therefore, $\frac{1}{x+1} \cdot \sum_{\tau_i \in pred(\tau_k)} \mu_{\max(0,\beta_i^+ - C_i)} < P(\Phi_k > x)$. Using Def. 9 and algebraic properties of the mean, we can rewrite this inequality as $\frac{\mu_{\Phi_k}}{x+1} < P(\Phi_k > x) = P(\Phi_k \geq x+1)$, which contradicts Markov's inequality [20], which states that $\mu_{\Phi_k}/(x+1) \geq P(\Phi_k \geq x+1)$ for all values of $x \geq 0$.

Case 3. $x < 0$. $P(\Phi_k > x) \leq 1$ is trivially true. □

Iteratively defining γ_k and completing the induction step. Here, we use Lems. 10 and 11 to compute the PDF of γ_k and prove the validity of the induction step. To do so, we first define a RV Δ_k^+ with PDF

$$f_{\Delta_k^+}(x) = \begin{cases} f_{\Psi_k^-}(-x) & x < 0 \\ 1 - \overline{F}_{\Psi_k^-}(0) - \overline{F}_{\Phi_k^+}(0) & x = 0 \\ f_{\Phi_k^+}(x) & x > 0. \end{cases} \tag{11}$$

Lemma 12. *For each node τ_k, $\Delta_k^+ \geq_1 \Delta_k$.*

Proof. **Case 1.** $x > 0$. In this case, by (11), $P(\Delta_k^+ > x) = P(\Phi_k^+ > x)$. However, by (5), $P(\Delta_k > x) = P(\Phi_k > x)$. Since Lem. 11 states that $\Phi_k^+ \geq_1 \Phi_k$, we have $\Delta_k^+ \geq_1 \Delta_k$.

Case 2. $x \leq 0$. In this case, $\Delta_k^+ > x$ gives rise to three possibilities: $x < \Delta_k^+ < 0$, or $\Delta_k^+ = 0$, or $\Delta_k^+ > 0$. By (11), the probability of these three events are $P(0 < \Psi_k^- < -x)$, $1 - P(\Psi_k^- > 0) - P(\Phi_k^+ > 0)$, and $P(\Phi_k^+ \geq 1)$ respectively. Thus, $P(\Delta_k^+ > x) = P(0 < \Psi_k^- < -x) + 1 - P(\Psi_k^- > 0) - P(\Phi_k^+ > 0) + P(\Phi_k^+ \geq 1)$. The first term on the rhs can be rewritten as $P(\Psi_k^- < -x) - P(\Psi_k^- \leq 0)$, and $P(\Psi_k^- \leq 0)$ can be rewritten as $1 - P(\Psi_k^- > 0)$. Cancelling like terms yields $P(\Delta_k^+ > x) = P(\Psi_k^- < -x)$. Through similar reasoning, $P(\Delta_k > x) = P(\Psi_k < -x)$ can be obtained from (5). It follows from Lems. 8 and 10 that $P(\Psi_k^- \leq -x) \geq P(\Psi_k \leq -x)$ holds. Thus, $\Delta_k^+ \geq_1 \Delta_k$. □

Using Δ_k^+, we can finally give the definition of γ_k.

Def. 10. *For each non-source node τ_k, $\gamma_k = \max(0, \Delta_k^+ + e_k)$.*

The following property, exploited in the theorem below, is a well-known closure property of stochastic orderings [37].

Property 9. *Let X_1 and X_2 be independent RVs and Y_1, Y_2 be another pair of independent RVs. If $Y_1 \geq_1 X_1$ and $Y_2 \geq_1 X_2$, then $Y_1 + Y_2 \geq X_1 + X_2$*

Theorem 3. *For each node τ_k, $\gamma_k \geq_1 \beta_k^+$.*

Proof. By expanding out the expression of Δ_k using (1) and Defs. 7 and 9, we see that Δ_k is expressed solely as a function of e_i for $\tau_i \in anc(\tau_k)$. Hence, by Asm. A1, Δ_k and e_k are independent. Δ_k^+ and e_k can similarly be shown to be independent by expanding Δ_k^+ using (6), (7), (9), and (11). Since $\Delta_k^+ \geq_1 \Delta_k$ due to Lem. 12, and $e_k \geq_1 e_k$ is trivially true, $\Delta_k^+ + e_k \geq_1 \Delta_k + e_k$ follows from Prop. 9. Thus, by Prop. 7, $\max(0, \Delta_k^+ + e_k) \geq_1 \max(0, \Delta_k + e_k)$, which implies $\gamma_k \geq_1 \beta_k^+$, by Def. 10 and the choice for β_k^+ (see Thm. 1). □

Thm. 3 implies that $P(\gamma_n > C_n)$ does indeed upper-bound the DAG drop rate. We can compute this probability by starting with the base case of $\gamma_1 = e_1$, and then inductively computing the PDF of $\gamma_i = \max(0, \Delta_i^+ + e_i)$ for each non-source task τ_i in the DAG. This can be done by using the PDF of γ_i for each $\tau_i \in pred(\tau_k)$ to first compute the PDFs of Ψ_k^- (resp. Φ_k^+) using (6) and (7) (resp. (9)). Ψ_k^- and Φ_k^+ can then be used to compute the PDF of Δ_k^+ using (11). Finally, γ_k can be computed by taking the PDFs of Δ_k^+ and e_k and applying Props. 3 and 4 to obtain the PDF of $\gamma_k = \max(0, \Delta_k^+ + e_k)$.

V. Experimental Evaluation

In this section, we experimentally compare the theoretical and experimental DAG drop rates of our proposed budgeting approach with the naïve strategy of aborting a DAG when just one of its nodes overruns its budget (as in [2]). These comparisons were performed by examining randomly generated DAGs, with the experimental drop rate obtained from a simulated four-core system using a G-EDF scheduler. In these experiments, we sought to **(i)** assess how our approach improves upon the naïve method analytically, **(ii)** demonstrate that our analytical bounds are correct, and **(iii)** determine the best method for selecting preferred predecessors.

Random DAG generation. In order to generate an n-node DAG G with a single source and sink, we first generated a random DAG G' with $n-2$ nodes using the *Erdős-Rényi method* [13]. Under this method, an edge is generated between each τ_i and τ_j with a specified probability p, with the edge directed from the lower-indexed task to the higher-indexed one. Using G', we then generated G by connecting a source node to all nodes in G' with indegree 0 and connecting all nodes in G' with outdegree 0 to a sink node.

Node parameter generation. For each DAG, we randomly generated the parameters of each node τ_i. To determine the

pWCET PDF for each τ_i, we assumed a type-1 Gumbel distribution with a mean of 5.0ms and a standard deviation of 2.0.[1] The reservation budget C_i was then set as the 99.9^{th} percentile value of the generated pWCET e_i. Finally, the period T and relative deadlines D_i were set to a large value of $(50n)\text{ms}$ so that the DAG was trivially schedulable (our focus here is dealing with node overruns, not ensuring schedulability).

Theoretical drop-rate evaluation. We evaluated the theoretical drop rate of each generated DAG using the analysis presented in Sec. IV. Additionally, unless otherwise specified, the drop rate was calculated from the upper-bounding method discussed in Sec. IV-C to avoid unnecessary independence assumptions and infeasible runtimes. This was then compared to the theoretical drop rate of the same generated DAG using the naïve budget-management strategy. In the naïve strategy, a DAG invocation completes (is not aborted) only when no node invocation overruns its budget. Therefore, since we set C_i as the 99.9^{th} percentile value of the generated pWCET, a DAG invocation is dropped with probability $1 - 0.999^n$.

Experimental drop-rate evaluation. We evaluated the experimental drop rate of each generated DAG by simulating the execution of each DAG over 10,000 invocations. We chose to select the available job with the maximum number of out edges as our job selection policy for Rules. B2.2, H2.1, and H2.2. We did this to avoid scenarios where an overrunning node blocks the execution of multiple subsequent jobs. For schedules of both our proposed budget-management strategy and the naïve one, we calculated the experimental drop rate from the percentage of dropped DAG invocations over the 10,000 simulated DAG invocations.

Experiment 1. In the first experiment, we generated 1,000 DAGs with n ranging from 50 to 500 and $p = 0.05$. Intuitively, a low value of p indicates a low-density DAG, which better represents real-world applications [17]. The results of this experiment are plotted in Fig. 7(a). These results support the following observation.

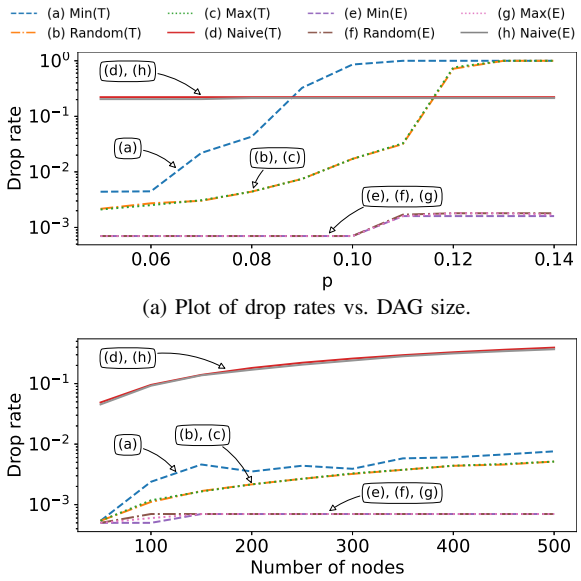
Observation 4. *With $p = 0.05$, our approach resulted in significantly lower drop rates both analytically and experimentally compared to the naïve approach at all DAG sizes.*

Experiment 2. In this experiment, we generated 1,000 250-node DAGs with p varying from 0.05 to 0.14. This was done to explore the effectiveness of our techniques on denser DAGs. The results of this experiment are plotted in Fig. 7(b). These results support the following observations.

Observation 5. *Our approach resulted in much lower experimental drop rates than the naïve approach for all p values.*

The discrepancy between the analytical and experimental drop rates was likely due to various sources of pessimism in our drop-rate analysis. First, in Def. 9, we assumed that no other reservations can help a job J_k complete jobs in $dep(J_k)$.

[1]The Gumbel distribution is often used to represent measurement-based probabilistic WCETs [14].



(a) Plot of drop rates vs. DAG size.

(b) Plot of drop rates vs. p.

Fig. 7: Plots for Exps. 1 and 2. Min, Random, and Max are the three policies for assigning preferred successors (see Sec. III). The suffix "T" (resp., "E") indicates theoretical (resp., experimental) drop rates. Both plots share the same legend.

Second, in Def. 7, we assumed that only a small amount of slack can benefit the analysis. Third, we used the pessimistic Markov Inequality to upper-bound the PDF of Φ_i.

Observation 6. *Analytically, our approach resulted in significantly lower drop rates for smaller values of p, but was worse than the naïve approach for larger p values.*

This was likely because larger p values increase the likelihood of certain DAG "features" (*e.g.*, a node with multiple predecessors) that present analysis difficulties. Such features exacerbate the pessimism of Def. 9 and the Markov Inequality.

Observation 7. *Using min degree to select preferred predecessors yielded extremely poor theoretical drop rates.*

This was likely because min degree discourages nodes with many predecessors—nodes most affected by analysis pessimism—from being able to benefit from slack reallocation.

Experiment 3. We ran an additional experiment to help elucidate the value, and the price, of eliminating the pessimism in using the Markov Inequality in Lem. 11. To this end, we generated five DAGs with $n = 7$ and $p = 0.1$. The maximum observed drop rate for these graphs was 0.00552, and the corresponding analytical drop-rate bound computed using Thm. 2 (resp., the method in Sec. IV-D) was 0.00613 (resp., 0.00698). However, using Thm. 2, computing these bounds for all graphs took three days on an eight-core machine, while using the method in Sec. IV-D, it took less than 1 second.

VI. Related Work

Prior DAG-related work has mostly focused on deriving response-time bounds and schedulability tests assuming known WCETs [1], [3], [4], [17], [23], [24], [27], [29], [33], [35], [36], [41], [43]. Reservation-based DAG scheduling has been considered in the context of federated scheduling where a reservation is provided for an entire DAG invocation to meet its deadline [5], [6], [26], [40]. However, this work assumes that true WCETs are known. Prior work on overrun management for real-time systems has focused on independent tasks on uniprocessors [11], [19], [28] or multiprocessors [31], [34] and mode switching in *mixed-criticality systems* [10], [12], [25], [30]. Slack reallocation has also been considered for independent tasks in the context of *mixed-criticality systems* [7], [16], [22], [32]. Recent work on time partitioning for graph-based tasks that require accelerator access has considered budget overruns of node invocations by aborting entire DAG invocations [2].

VII. Conclusion

We have presented the first ever holistic budget-management strategy for DAG-based task system. This strategy enables low DAG drop rates in the presence of node-budget overruns, and can reallocate budgets and slack within a DAG. We have also given a probabilistic drop-rate analysis to analytically evaluate our proposed approach. In experiments presented herein, our work yielded significantly smaller theoretical drop rates for less complex DAGs, and smaller experimental drop rates for all DAGs, compared to the naïve DAG budgeting approach.

In future work, we hope to reduce the analytical pessimism for complex DAGs. We also plan to investigate the application-level implications of occasionally needing to drop work. Additionally, we plan to devise budgeting techniques for DAGs that share resources such as hardware accelerators, as commonly used in the AI applications that motivate this work. Finally, our task model allows successive invocations of the same DAG to overlap in time arbitrarily. It would be interesting to consider refinements of this model, such as the *restricted parallelism (rp)* model [3], which allows a specifiable extent of overlap. Our drop-rate analysis can be extended to apply to the rp model via fairly mechanical adjustments, but some of our probabilistic upper bounds are somewhat pessimistic under that model, so further work is warranted. Note that for DAGs subject to hard real-time schedulability tests that preclude the overlap of successive DAG invocations, the issue of overlap becomes irrelevant and our analysis is entirely applicable.

References

[1] S. Ahmed and J. Anderson, "Exact response-time bounds of periodic DAG tasks under server-based global scheduling," in *RTSS*, 2022, pp. 447–459.

[2] T. Amert, Z. Tong, S. Voronov, J. Bakita, F. D. Smith, and J. Anderson, "Timewall: Enabling time partitioning for real-time multi-core+accelerator platforms," in *RTSS*, 2021, pp. 455–468.

[3] T. Amert, S. Voronov, and J. Anderson, "OpenVX and real-time certification: The troublesome history," in *RTSS*, 2019, pp. 312–325.

[4] S. Baruah, "Improved multiprocessor global schedulability analysis of sporadic DAG task systems," in *ECRTS*, 2014, pp. 97–105.

[5] ——, "The federated scheduling of constrained-deadline sporadic DAG task systems," in *DATE*, 2015, pp. 1323–1328.

[6] ——, "Federated scheduling of sporadic DAG task systems," in *ISORC*, 2015, pp. 179–186.

[7] I. Bate, A. Burns, and R. I. Davis, "A bailout protocol for mixed criticality systems," in *ECRTS*, 2015, pp. 259–268.

[8] T. Blaß, A. Hamann, R. Lange, D. Ziegenbein, and B. Brandenburg, "Automatic latency management for ROS 2: Benefits, challenges, and open problems," in *RTAS*, 2021, pp. 264–277.

[9] A. Burns, G. Bernat, and I. Broster, "A probabilistic framework for schedulability analysis," in *EMSOFT*, 2003, pp. 1–15.

[10] A. Burns, R. Davis, S. Baruah, and I. Bate, "Robust mixed-criticality systems," *IEEE Trans. Computers*, vol. 67, no. 10, pp. 1478–1491, 2018.

[11] M. Caccamo, G. Buttazzo, and L. Sha, "Handling execution overruns in hard real-time control systems," *IEEE Trans. Computers*, vol. 51, no. 7, pp. 835–849, 2002.

[12] G. Chen, N. Guan, D. Liu, Q. He, K. Huang, T. Stefanov, and W. Yi, "Utilization-based scheduling of flexible mixed-criticality real-time tasks," *IEEE Trans. Computers*, vol. 67, no. 4, pp. 543–558, 2018.

[13] D. Cordeiro, G. Mounié, S. Perarnau, D. Trystram, J. Vincent, and F. Wagner, "Random graph generation for scheduling simulations," in *SIMUTools*, 2010.

[14] R. Davis and L. Cucu-Grosjean, "A survey of probabilistic timing analysis techniques for real-time systems," *Leibniz Trans. Embedded Syst.*, vol. 6, no. 1, pp. 03:1–03:60, 2019.

[15] J. Erickson and J. Anderson, "Response time bounds for G-EDF without intra-task precedence constraints," in *OPODIS*, 2011, pp. 128–142.

[16] T. Fleming, "Extending mixed criticality scheduling," 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:15603980

[17] J. C. Fonseca, G. Nelissen, and V. Nélis, "Schedulability analysis of DAG tasks with arbitrary deadlines under global fixed-priority scheduling," *Real Time Syst.*, vol. 55, no. 2, pp. 387–432, 2019.

[18] C. M. Fortuin, J. Ginibre, and P. W. Kasteleyn, "Correlation inequalities on some partially ordered sets," *Comm. in Math. Phys.*, vol. 22, no. 2, pp. 89 – 103, 1971.

[19] M. Gardner and J. Liu, "Performance of algorithms for scheduling real-time systems with overrun and overload," in *ECRTS*, 1999, pp. 287–296.

[20] B. K. Ghosh, "Probability inequalities related to Markov's Theorem," *Am. Stat.*, vol. 56, no. 3, pp. 186–190, 2002.

[21] G. Grimmett, *Percolation*. Springer Berlin, Heidelberg, 1999, ch. 2, p. 34.

[22] J.-J. Han, X. Tao, D. Zhu, H. Aydin, Z. Shao, and L. T. Yang, "Multicore mixed-criticality systems: Partitioned scheduling and utilization bound," *IEEE Trans. on Comput.-Aided Des. Integr. Circuits Syst.*, vol. 37, no. 1, pp. 21–34, 2018.

[23] Q. He, N. Guan, M. Lv, X. Jiang, and W. Chang, "Bounding the response time of DAG tasks using long paths," in *RTSS*, 2022, pp. 474–486.

[24] Q. He, M. Lv, and N. Guan, "Response Time Bounds for DAG Tasks with Arbitrary Intra-Task Priority Assignment," in *ECRTS*, 2021, pp. 8:1–8:21.

[25] B. Hu, K. Huang, P. Huang, L. Thiele, and A. Knoll, "On-the-fly fast overrun budgeting for mixed-criticality systems," in *EMSOFT*, 2016, pp. 25:1–25:10.

[26] X. Jiang, N. Guan, H. Liang, Y. Tang, L. Qiao, and W. Yi, "Virtually-federated scheduling of parallel real-time tasks," in *RTSS*, 2021, pp. 482–494.

[27] J. Li, K. Agrawal, C. Lu, and C. Gill, "Analysis of global EDF for parallel tasks," in *ECRTS*, 2013, pp. 3–13.

[28] C. Lin and S. Brandt, "Improving soft real-time performance through better slack reclaiming," in *RTSS*, 2005, pp. 12 pp.–421.

[29] C. Liu and J. Anderson, "Supporting soft real-time DAG-based systems on multiprocessors with no utilization loss," in *RTSS*, 2010, pp. 3–13.

[30] D. Liu, J. Spasic, N. Guan, G. Chen, S. Liu, T. Stefanov, and W. Yi, "EDF-VD scheduling of mixed-criticality systems with degraded quality guarantees," in *RTSS*, 2016, pp. 35–46.

[31] M. Liu, M. Behnam, S. Kato, and T. Nolte, "A server-based approach for overrun management in multi-core real-time systems," in *ETFA*, 2014, pp. 1–10.

[32] M. S. Mollison, J. P. Erickson, J. H. Anderson, S. K. Baruah, and J. A. Scoredos, "Mixed-criticality real-time scheduling for multicore systems," in *ICCIT*, 2010, pp. 1864–1871.

[33] A. Parri, A. Biondi, and M. Marinoni, "Response time analysis for G-EDF and G-DM scheduling of sporadic DAG-tasks with arbitrary deadline," in *RTNS*, 2015, pp. 205–214.

[34] R. Pellizzoni and M. Caccamo, "M-CASH: A real-time resource reclaiming algorithm for multiprocessor platforms," *Real Time Syst.*, vol. 40, no. 1, pp. 117–147, 2008.
[35] M. Qamhieh, F. Fauberteau, L. George, and S. Midonnet, "Global EDF scheduling of directed acyclic graphs on multiprocessor systems," in *RTNS*, 2013, pp. 287–296.
[36] A. Saifullah, D. Ferry, C. Lu, and C. Gill, "Real-time scheduling of parallel tasks under a general DAG model," *IEEE Trans. Parallel Distributed Syst.*, 2012.
[37] M. Shaked and J. G. Shanthikumar, *Stochastic Orders*. Springer, 2007, pp. 5–6.
[38] Z. Tong, S. Ahmed, and J. Anderson, "Overrun-resilient multiprocessor real-time locking," in *ECRTS*, 2022, pp. 9:1–9:23.
[39] ——, "Holistically budgeting processing graphs (longer version)," 2023. [Online]. Available: http://jamesanderson.web.unc.edu/papers/
[40] N. Ueter, G. Brüggen, J. Chen, J. Li, and K. Agrawal, "Reservation-based federated scheduling for parallel real-time tasks," in *RTSS*, 2018, pp. 482–494.
[41] K. Wang, X. Jiang, N. Guan, D. Liu, W. Liu, and Q. Deng, "Real-time scheduling of DAG tasks with arbitrary deadlines," *ACM Trans. Design Autom. Electr. Syst.*, vol. 24, no. 6, pp. 66:1–66:22, 2019.
[42] R. Wilhelm, "Real time spent on real time," in *RTSS*, 2020, pp. 1–2.
[43] K. Yang, M. Yang, and J. Anderson, "Reducing response-time bounds for DAG-based task systems on heterogeneous multicore platforms," in *RTNS*, 2016, pp. 349–358.